Docket No. 006570.P009
Express Mail No. EV339914118US

UNITED STATES PATENT APPLICATION

FOR

CLUSTER ARCHITECTURE HAVING A STAR TOPOLOGY WITH CENTRALIZED SERVICES

Inventors:

Hans-Christoph Rohland
Frank Kilian

Prepared By:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard, 7th Floor
Los Angeles, California 90025-1030
(310) 207-3800

STAR TOPOLOGY WITH CENTRALIZED SERVICES HAVING A CLUSTER ARCHITECTURE

BACKGROUND

Field of the Invention

[0001] The invention relates to cluster architecture. More specifically, the invention relates to a star topology cluster architecture for application servers.

Background

[0002] Traditionally, each instance in a cluster of application servers maintains a direct communication link with each other instance. In the cluster, as used herein, instance refers to a unit in the cluster, which can be started, stopped, and monitored separately from other units in the cluster. As the cluster becomes increasingly large, the overhead associated with maintaining the communication link from each instance to every other instance becomes quite burdensome. Resulting performance problems have created a real constraint on cluster size. Additionally, to maintain the homogeneity of the cluster, large amounts of data are replicated and passed around the cluster over these myriad communication links. This overhead severely impacts the scalability of such cluster architectures.

SUMMARY

[0003] A cluster of architecture having a star topology with a central services node at its center is described. Application server instances are organized in the star topology with the central services node at its center. The central services node may include services such as a messaging server for interinstance cluster communications. A locking server may also be provided to provide cluster wide locking to facilitate changes and updates of data within the cluster or to synchronize concurrent processes. A database may also be shared by all instances in the cluster, thereby reducing the need for data replication. In one embodiment, the message server has no persistent state. In such an embodiment, if the message server fails, it can merely be restarted without any state recovery requirement.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that references to "an" or "one" embodiment in this disclosure are not necessarily to the same embodiment, and such references mean at least one.

[0005] **Figure 1** is a block diagram of a system employing a cluster architecture of one embodiment of the invention.

[0006] **Figure 2** is a fundamental modeling concept (FMC) diagram of dispatcher/server process interaction.

[0007] **Figure 3** is a flow diagram of a cluster establishment in one embodiment of the invention.

[0008] **Figures 4** and **5** illustrate one exemplary startup method and framework, respectively.

DETAILED DESCRIPTION

[0009] **Figure 1** is a block diagram of a system employing a cluster architecture of one embodiment of the invention. A cluster includes central services node 102 which is organized in star topology with one or more application server instances 104 (104-1 to 104-M) with the central services node 102 at the center of the star topology. Application server instances 104 received requests from one or more clients for example, via web browsers 108 (108-1 to 108-N) over a distributed network 106. In one embodiment, the network 106 may be a distributed network such as the Internet. In one embodiment, requests from the client web browser 108 may be transmitted using hypertext transfer protocol (HTTP). In one embodiment, the cluster is a Java 2 enterprise edition (J2EE) cluster.

[0010] As previously noted, as used herein, each "instance" is a unit in the cluster, which can be started, stopped and monitored separately. In one embodiment, each instance runs on a physical server, but more than one instance may run on a single physical server. In one embodiment, an instance may be identified within the cluster by a system identification number and an instance number. In one embodiment, the central services node is an example of a J2EE instance.

[0011] An instance typically contains at least one server process 124. More commonly, an instance includes a dispatcher 122 and several server processes 124. The dispatcher 122 and the server process 124 are described in greater detail below in connection with **Figure 2**. It is also contemplated that more than one dispatcher may reside in a single instance.

[0012] In one embodiment, central services node 102 includes a message server 110, lock server 112 and a shared database 114. The message server 110 maintains a client list 116. The client list 116 includes a listing of the dispatchers 122 and server processes 124 of the cluster. Client list 116 should not be confused with a list of client web browsers 108, but rather, is a list of the clients of the message server. The message server also maintains a service list 118, which lists the services available within the cluster such as, for example, an HTTP service.

[0013] Message server 110 is responsible for interinstance communication. For example, if a server process 124 of instance 104-1 wishes to send a message 126 to instance 104-M, a message 126 is sent via the message server 110 which provides the infrastructure for interinstance communication. In one embodiment, each server process 124 and each dispatcher 122 has a link through which it can communicate with the message server, and through which the message server may send messages notifying of events within the cluster. Message server 110 also supplies a dispatcher 122 with information to facilitate load balancing between various instances in the cluster. As noted above, the message server 110 also provides notification of events that arise within a cluster, for example, failure or shutdown of an instance or when a service is started or stopped. Because the message server may represent a single point of failure in the cluster, it should support failover to be effectively used in high availability systems. To that end, in one embodiment, the message server has no persistent state such that if the message server fails, it need merely be restarted and then re-register instances in the cluster without performing any state recovery procedures.

[0014] Lock server 112 is used for internal synchronization of the cluster. Applications may lock objects and then release the objects subsequently. This is particularly true in the context of updating the cluster, with for example, deployment of a new application. Lock server 112 maintains a lock table 120 in

which it manages logical database locks to prevent inconsistent modifications of, for example, database 114. In one embodiment, the lock table is maintained in main memory (not shown). In one embodiment, lock table 120 maps logical locks to data that is contained in database 114. The lock sever 112 may also represent a single point of failure within the cluster. For high availability systems, a replication server may be provided which maintains a real time replica of the state of the lock server 112. After lock server failure, the lock server 112 may be restarted with the replication of the state from the replication server.

[0015] **Figure 2** is a fundamental modeling concept (FMC) diagram of dispatcher/server process interaction. Initially, the client 208 requests connection from the connection request handler 202 using, for example, HTTP. A load balancer 204 provides an indication of which server process 124 is going to handle the client response request from the connection request handler 202. In one embodiment, load balancer 204 uses a weighted round robin procedure. The connection request handler 202 initializes a connection object, which is then assigned to the client 208 after which the client will have connection to the connection manager 206 going forward. Client 108-1 and 108-M are shown with connections to connection manager 206 already in place. Once the connection request handling 202 creates the connection object and establishes the connection, connection manager 206 identifies what session level services 210, such as HTTP or RMI (remote method in location) for example, will be used. All requests from the same client are sent to the same server process 124 thereafter. The connection manager 206 then forwards the request including the session level services to the communication handler 212 where they may be queued in request queue 214 until thread manager 213 provides a worker thread 215 to service the requests to be sent to the server process 124.

[0016] Request to the server process 124 in one embodiment, may be sent using transmission control protocol/internet protocol (TCP/IP). Communications handler 218 in the server process receives the request from the dispatcher 122 and queues it in the request queue 226 until thread manager 222 provides a worker thread 224 to service the request. Session level services210, which may be the same as those assigned in the dispatcher node 122, are applied and the thread attempts to service the request using the

application/application level service 220. Because each server processes is multi-threaded, a large number of requests may be serviced simultaneously. The response to the request is sent back through the dispatcher 122 over the connection established within the connection manager 206 to the client 108.

[0017] **Figure 3** is a flow diagram of a cluster establishment in one embodiment of the invention. At block 302, the central services node (CSN) is started. In one embodiment, the central services are provided by the central services node started on a physical server with its own system number and the system identification number of the whole system. Once the central services node is up and running, additional instances are started at block 304. One manner in which additional instances may be started is described with reference to **Figures 4** and **5** below. The dispatcher and server processes of the various instances register with the central services node at block 306. In one embodiment, a message server in the central services node maintains a list of connected clients including a system ID and instance number for each dispatcher and server process running in the cluster. At block 308, the central services node notifies all current cluster registrants as each additional instance joins the cluster. This may include notifying the cluster constituents of services available within the cluster start up, including any new services deployed as a result of additional instances during a cluster or additional application deployment.

[0018] A determination is made at decision block 309 if a failure has occurred in the central services node. Failure may be the result of, for example, the message server failing, lock server failure or database failure. If a failure has occurred, a determination is made at block 310 if the database has failed. If it has, the database is recovered according to procedures established by the database vendor at block 312. If the database has not failed, or after recovery, a determination is made at decision block 314, if the central services node lock server has failed. If it has, lock server is restarted using a replica from the replication server at block 316. A determination is then made at decision block 318 when the message server has failed. If the message server has failed, the message server is restarted at block 420. Once the message server has restarted, registration of the instances can recommence at block 306 as previously described. If the central services node has not failed, failure is

awaited at block 309. While Figure 3 depicts a flow diagram, in various embodiments certain operations may be conducted in parallel or in a different order than depicted. Accordingly, different ordering and parallelism are within the scope of the invention.

[0019] One embodiment of the invention employs a unique startup framework for starting and stopping the various server instances within the cluster. **Figures 4** and **5** illustrate one exemplary startup method and framework, respectively. The startup framework 900 includes startup and control logic 902 and bootstrap logic 901. In one embodiment, the startup and control logic 902 provides the central point of control for the instance 104 and for all processes 903 executed within the servers and dispatchers of the instance 104. For example, the instance startup procedure described herein is performed under the control of the startup and control logic 902.

[0020] Turning to the method in **Figure 4**, at 800, the startup and control program 902 launches the bootstrap logic 901 to initiate startup of the instance (e.g., in response to a startup command entered by a network administrator). As illustrated in **Figure 5**, the bootstrap logic 901 is comprised of bootstrap binaries 513 and is configured based on bootstrap configuration parameters 512 stored within the configuration data hierarchy 420 of the central database 114. Thus, if necessary, the bootstrap logic 901 may be modified/updated at a single, central location and subsequently distributed to servers/instances upon request.

[0021] At 802, the bootstrap logic retrieves up-to-date configuration data 420 from the central database 114 including the layout of the instance 104 (e.g., identifying the servers and dispatchers to be started) and the parameters/ arguments to be used for each server and dispatcher within the instance 104. In one embodiment, the bootstrap logic 901 uses this information to construct a description of the instance, which it provides to the startup and control logic 902 in the form of an "Instance Properties" data object. In one embodiment, the Instance Properties data object is simply a text file containing a list of servers/dispatchers and associated parameters which the startup and control logic parses to determine the instance layout and settings. However, various alternate data formats may be employed for the Instance Properties file while

still complying with the underlying principles of the invention (e.g., such as the "Extensible Markup Language" or "XML" format).

[0022] At 804, using the instance layout and configuration data from the Instance Properties data object, the startup and control logic 902 builds a list of servers and/or dispatchers to be started, including an indication of the specific instance processes to be started on each server/dispatcher. In one embodiment of the invention, prior to starting each server/dispatcher identified in the list, the startup and control logic 902 launches node-specific bootstrap logic 905, at 806, to synchronize the binaries and configuration settings on each server and/or dispatcher. For example, if the instance 104 was inoperative for a period of time, the global and/or server/dispatcher-specific binaries and configuration settings 900 may have changed within the central database 114. Accordingly, in one embodiment, when the node-specific bootstrap logic 905 is launched, it compares the binaries and configuration settings stored in the local file system of the server/dispatcher being started to the binaries and configuration settings 900 stored in the central database 114. In one embodiment, the comparison is performed between an index of the data stored locally on the server/dispatcher and an index of the data stored within the central database 420 (e.g., an index built from the hierarchical structure illustrated in **Figure 5**).

[0023] Regardless of how the comparison is performed, if the binaries and/or configuration settings stored within the local file system 904 of the dispatcher/ server are out-of-date, then the current binaries and/or configuration settings 900 are retrieved from the central database 114 and stored within the local file system 904 of the dispatcher/server. In one embodiment, only the binaries and/or configuration settings which are new are copied to the local file system 904, thereby conserving network bandwidth and server resources.

[0024] Once synchronization is complete, at 808, the startup and control logic executes the processes on each of the servers using arguments/parameters included within the Instance Properties data object. At 810, the startup and control logic initializes the service framework and services on the servers/ dispatchers within the instance 104. For example, the service framework and services are the J2EE service framework and J2EE services,

respectively, in a Java environment. However, various other types of services/frameworks may be employed while still complying with the underlying principles of the invention.

[0025] Embodiments of the invention may include various operations as set forth above. The operations may be embodied in machine-executable instructions which cause a general-purpose or special-purpose processor to perform certain operations. Alternatively, these operations may be performed by specific hardware components that contain hardwired logic for performing the operations, or by any combination of programmed computer components and custom hardware components.

[0026] Elements of the present invention may also be provided as a machine-readable medium for storing the machine-executable instructions. The machine-readable medium may include, but is not limited to, flash memory, optical disks, CD-ROMs, DVD ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, propagation media or other type of machine-readable media suitable for storing electronic instructions. For example, the present invention may be downloaded as a computer program which may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection).

[0027] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however be evident that various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.